09-05-07

JFW

Attorney's Docket No. 042933/298965

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Inventor:	Nativadade Lobo	Confirmation No.: 5615
Appl. No.:	09/625,201	
Filed:	July 21, 2000	
For:	PULSE SHAPING WHICH COMPENSATES FOR COMPONENT DISTORTION	

BOX ISSUE FEE
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

SUBMITTAL OF PRIORITY DOCUMENT

To complete the requirements of 35 U.S.C. § 119, enclosed is a certified copy of Great Britain priority Application No. 9804600.6, filed March 5, 1998.

Respectfully submitted,

Cory C. Davis
Registration No. 59,932

Customer No. 00826
Alston & Bird LLP
Bank of America Plaza
101 South Tryon Street, Suite 4000
Charlotte, NC 28280-4000
Tel Charlotte Office (704) 444-1000
Fax Charlotte Office (704) 444-1111

"Express Mail" mailing label number EV521113148US
Date of Deposit September 4, 2007

I hereby certify that this paper or fee is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to: BOX ISSUE FEE, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450

Tamara Stevens

tellectual
operty
ffice

ivity and Innovation

Concept House
Cardiff Road
Newport
South Wales
NP10 8QQ

I, the undersigned, being an officer duly authorised in accordance with Section 74(1) and (4) of the Deregulation & Contracting Out Act 1994, to sign and issue certificates on behalf of the Comptroller-General, hereby certify that annexed hereto is a true copy of the documents as originally filed in connection with patent application GB9804600.6 filed on 5 March 1998.

In accordance with the Patents (Companies Re-registration) Rules 1982, if a company named in this certificate and any accompanying documents has re-registered under the Companies Act 1980 with the same name as that with which it was registered immediately before re-registration save for the substitution as, or inclusion as, the last part of the name of the words "public limited company" or their equivalents in Welsh, references to the name of the company in this certificate and any accompanying documents shall be treated as references to the name with which it is so re-registered.

In accordance with the rules, the words "public limited company" may be replaced by p.l.c., plc, P.L.C. or PLC.

Re-registration under the Companies Act does not constitute a new legal entity but merely ompany to certain additional company law rules.

Signed

Dated 21 August 2007

Patents Form 1/77

Patents Act 1977
(Rule 16)

THE PATENT OFFICE
A
5 MAR 1998
RECEIVED BY FAX

The Patent Office

9
05MAR98 E343134-1 D02716
P0177700 25.00 - 9804600.6

Request for grant of a patent

(See the notes on the back of this form. You can also get an explanatory leaflet from the Patent Office to help you fill in this form)

The Patent Office

Cardiff Road
Newport
Gwent NP9 1RH

1. Your reference

PAT 98005 GB

2. Patent **9804600.6**
(The Patent Office will fill in this part)

05 MAR 1998

3. Full name, address and postcode of the or of each applicant *(underline all surnames)*

NOKIA MOBILE PHONES LIMITED
KEILALAHDENTIE 4
02150 ESPOO
FINLAND

Patents ADP number *(if you know it)*

5911995004

If the applicant is a corporate body, give the country/state of its incorporation

4. Title of the invention

A CDMA MODULATOR/DEMODULATOR

5. Name of your agent *(if you have one)*

"Address for service" in the United Kingdom to which all correspondence should be sent *(including the postcode)*

MRS HELEN LOUISE HAWS
NOKIA MOBILE PHONES
PATENT DEPARTMENT
ST GEORGES COURT
ST GEORGES ROAD
CAMBERLEY
SURREY GU15 3QZ UK

Patents ADP number *(if you know it)*

6945539001

6. If you are declaring priority from one or more earlier patent applications, give the country and the date of filing of the or of each of these earlier applications and *(if you know it)* the or each application number

Country	Priority application number *(if you know it)*	Date of filing *(day / month / year)*

7. If this application is divided or otherwise derived from an earlier UK application, give the number and the filing date of the earlier application

Number of earlier application	Date of filing *(day / month / year)*

8. Is a statement of inventorship and of right to grant of a patent required in support of this request? *(Answer 'Yes' if:*

a) any applicant named in part 3 is not an inventor, or
b) there is an inventor who is not named as an applicant, or
c) any named applicant is a corporate body.
See note (d))

YES

I certify this to be a true copy.

G D Court.

Acting for Comptroller

Patents Form 1/77

Patents Form 1/77

9. Enter the number of sheets for any of the following items you are filing with this form. Do not count copies of the same document

Continuation sheets of this form	
Description	28
Claim(s)	
Abstract	
Drawing(s)	

10. If you are also filing any of the following, state how many against each item.

Priority documents	
Translations of priority documents	
Statement of inventorship and right to grant of a patent *(Patents Form 7/77)*	
Request for preliminary examination and search *(Patents Form 9/77)*	
Request for substantive examination *(Patents Form 10/77)*	
Any other documents *(please specify)*	

11. I/We request the grant of a patent on the basis of this application.

Signature H L Haws Date 5/3/98

H L HAWS - AGENT FOR THE APPLICANT

12. Name and daytime telephone number of person to contact in the United Kingdom: Miss K Jeffery - 01276 419538

Warning

After an application for a patent has been filed, the Comptroller of the Patent Office will consider whether publication or communication of the invention should be prohibited or restricted under Section 22 of the Patents Act 1977. You will be informed if it is necessary to prohibit or restrict your invention in this way. Furthermore, if you live in the United Kingdom, Section 23 of the Patents Act 1977 stops you from applying for a patent abroad without first getting written permission from the Patent Office unless an application has been filed at least 6 weeks beforehand in the United Kingdom for a patent for the same invention and either no direction prohibiting publication or communication has been given, or any such direction has been revoked.

Notes

a) If you need help to fill in this form or you have any questions, please contact the Patent Office on 0645 500505.

b) Write your answers in capital letters using black ink or you may type them.

c) If there is not enough space for all the relevant details on any part of this form, please continue on a separate sheet of paper and write "see continuation sheet" in the relevant part(s). Any continuation sheet should be attached to this form.

d) If you have answered 'Yes' Patents Form 7/77 will need to be filed.

e) Once you have filled in the form you must remember to sign and date it.

f) For details of the fee and ways to pay please contact the Patent Office.

CDMA Patent

The process at the transmitter of CDMA is roughly

BitStream –> Frame Builder –>**GoldCode Encoding** –> Modulator **[Using LookUp Table]**
D/A –> Filter[Switched Capacitor,**Clock Frequency**] –> Filter[**Resistors,Capasitors**] –> Transmitter

The corresponding process for CDMA receiver is

Complex Number Seq –> **CDMA Process** –> Channel Estimation –> Demodulation [Using Viterbi] –> BitStream

These process are in fact similar to GSM so we could build a dual mode phone. We describe the GSM processes

The transmitter processes are listed below

BitStream –> Frame Builder –> Modulator **[Using LookUp Table]**
D/A –> Filter[Switched Capacitor,**Clock Frequency**] –> Filter[**Resistors,Capasitors**] –> Transmitter

At the receiver we have the reverse process

Complex Number Seq –> **Multiplying incomming stream by i^n** –> **BitSinchronisation** –> Channel Estimation –> Demodulation **[Using Viterbi]** –> BitStream

Note
1)The mode dependent process are highlighted.
2) The same filter can be used in both the cases if the clock frequencies are selected with care.
3)The timing issues, and frequency dependent items like antennas have been ignored in this paper.

4)In this paper we use many different pulses depending on the bandwidth, chip rate, required and we trade off the non constant amplitude. This is the key inovation (to get higher bit rates).
The selection of bitrate is just a matter of engineering design involving BER. Power Amplifier efficiency and spectrum utilisation. We also point out that although in this paper we have designed a modulator from first principles, we would in pratice use the current Nokia Mobile Phones lookup table architecure [4T or 6T] obtaining the correct syclostatioary spectrum in each mode by the appropiate filter [Analog]. The Switched Capacitor will in pratice do most of the shaping common to both. This again is subject to design

5)The key inovative step is the nature of the function used to transform the gold code into the sequence used in the correlator by the function below. It is this function that we wish to protect in the patent. The transformation need only occur once,and the results stored, as the mobile can pre calculate the sequence d_i for $i = 0,1, .. N-1$ given the code $\{c_0, c_1, \ldots c_{N-1}\}$ when the code is asignment.

$y_i = (-1)^i$ for $i = 0, 1, 2, \ldots N-1$
Given code $\{c_0, c_1, \ldots c_{N-1}\}$ where N is the number of elements in the sequence .
$a_i = 1$ if $c_i = 1$; for $i = 0, \ldots N-1$
$a_i = -1$ if $c_i = 0$; for $i = 0, \ldots N-1$
$b_0 = a_0$:
$b_i = b_{i-1} + a_i$ for $i = 1, 2, \ldots N-1$;
$d_i = y_i\, i^{b_i}$ for $i = 0, 1, 2, \ldots N-1$ and
$i = \sqrt{(-1)}$

5)The function is behaviour is specified in the variable x4 in the CDMACoarseSynchroniser.
We also stress that while the bitstream is differentially encoded and the substitution used is $\{0 \to 1, 1 \to -1\}$ the substitution elsewhere used for the gold codes is $\{0 \to -1, 1 \to 1\}$

6)The demodulation in the CDMA mode does not need a Viterbi algorithm as is tequired by the GSM mode. Thus, most of the work is done by the gold code. during correlation.

7)There is nothing special about the Gold Code used in the example. Any gold like code will do as long as it posseses the usual properties of a gold code.

8). It is not necesary to have a long training sequence as in GSM, because the code is longer M-sequence, which is in fact the gsm training seq.

The function is behaviour is specified in the variable x4 in the CDMACoarseSynchroniser.
We also stress that while the bitstream is differentially encoded and the substitution used is $\{0 \to -1, 1 \to -1\}$ the substitution elsewhere used for the gold codes is $\{0 \to -1, 1 \to 1\}$

6)The demodulation in the CDMA mode does not need a Viterbi algorithm as is tequired by the GSM mode. Thus, most of the work is done by the gold code. during correlation.
7)There is nothing special about the Gold Code used in the example. Any gold like code will do as long as it posseses the usual properties of a gold code.

8) It is not necessary to have a long training sequence as in GSM, because all the work is done by the code

■ Dual Mode CDMA and GSM Operation

In this work we demonstrate how to build a dual mode phone using as much as possible the same hardware for both the systems.
Given the following process for GSM Transmitter

...Stream –> Frame Builder –> Modulator **[Using LookUp Table]**
D/A –> Filter[Switched Capacitor,**Clock Frequency**] –> Filter[**Resistors,Capasitors**] –> Transmitter

At the receiver we have the reverse process

Complex Number Seq –> Channel Estimation –> Demodulation **[Using Viterbi]** –> BitStream

The corresponding process at the CDMA transmitter is roughly

BitStream –> Frame Builder –>**GoldCode Encoding** –> Modulator **[Using LookUp Table]**
D/A –> Filter[Switched Capacitor,**Clock Frequency**] –> Filter[**Resistors,Capasitors**] –> Transmitter

The corresponding process for CDMA receiver is

Complex Number Seq –> **CDMA Process** –> Channel Estimation –> Demodulation **[Using Binary Decision]** –> BitStream

Note
1)The mode dependent process are highlighted.
2) The same filter can be used in both the cases if the clock frequencies are selected with care.
3)The timing issues, and frequency dependent items like antennas have been ignored in this paper.

4)In this paper we use many different pulses depending on the bandwidth, chip rate, required and we trade off the non constant amplitude. This is the key inovation (to get higher bit rates).
The selection of bitrate is just a matter of engineering design involving BER, Power Amplifier efficiency and spectrum utilisation. We also point out that although in this paper we have designed a modulator from first principles, we would in pratice use the current Nokia Mobile Phones lookup table architecure [based on a pulse width of 4T] or an architecure based on 6T pulse width obtaining the correct syclostatioary spectrum in each mode by the appropiate filter [Analog]. The Switched Capacitor will in pratice do most of the spectral shaping common to both modes. This again is standard to design

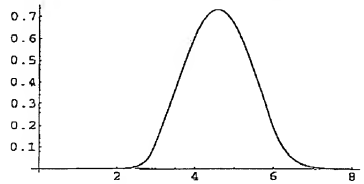
5)The key inovative step is the nature of the function used to transform the gold code into the sequence used in the correlator by the function below. It is this function that we wish to protect in the patent. The transformation need only occur once,and the results stored, as the mobile can pre calculate the sequence d_i for i = 0,1, .. N–1 given the code $\{c_0, c_1, \ldots c_{N-1}\}$ when the code is asignment.

$y_i = (-1)^i$ for $i = 0, 1, 2, \ldots N-1$
Given code $\{c_0, c_1, \ldots c_{N-1}\}$ where N is the number of elements in the sequence.
$a_i = 1$ if $c_i = 1$; for $i = 0, \ldots N-1$
$a_i = -1$ if $c_i = 0$; for $i = 0, \ldots N-1$
$b_0 = a_0$;
$b_i = b_{i-1} + a_i$ for $i = 1, 2, \ldots N-1$;
$d_i = y_i i^{b_i}$ for $i = 0, 1, 2, \ldots N-1$ and
$i = \sqrt{(-1)}$

```
In[1]:= Needs["RingFunctionsDiffEncoded`"]

In[2]:= Names["RingFunctionsDiffEncoded`*"]

Out[2]= {AllGoldSequences, AutocorrelationSequence,
  AutomorphismSigma, CrosscorrelationSequence, CyclicMultiplativeGroup,
  DropLeadingZeros, GoldSequence, GraeffeMethod, InitialConditions, MinimumPoly,
  ModuloMultplication, PolynomialMultiplication, PossibleDivisors, RingDivision,
  RingPower, SequenceGenerator, SpecifiedGoldSequences, TraceRepresentation,
  TupleRepresentation, TwoAdicExpansion, UnitsRing, ZeroPad, ZeroSequences, T, σ}

In[3]:= Needs["LaurentFunctions`"]

        RuleDelayed::rhs : Pattern t_ appears on the right-hand side of rule
          PhaseAngle[L_][t_] :> (PhaseAngle[L][t_] = Module[{x1, x2, x3, x4, x5, x6}, <<1>>]).

In[4]:= Needs["LaurentNotationTest`"]

        Needs::nocont : Context LaurentNotationTest` was not created when Needs was evaluated.
```

Information on the functions used can be obtained using help.

```
In[5]:= Names["LaurentFunctions`*"]

Out[5]= {AKN, AlphaKI, ANKInitialStateSetUp, BT, FiltPulse, h, hFiltered, InitialState, J,
  LaurentC, LaurentLK, LaurentS, M, ModulatingPulse, ModulationIndex, Modulator,
  NumberOfCurves, PhaseAngle, PhaseAngleFast, Receiver, ReceiverProper, S,
  SamplingInterval, StartingQuadrant, SyncSample, T, C, Φ, ψ}

In[6]:= T := 3/812500
        BT := 0.3

In[8]:= ModulationIndex := 1/2

In[9]:= << ModulatorData.m;

In[10]:= << OptimalPulseShaped.m;

In[11]:= Plot[OptPulse[L][0][t], {t, 0, 8}]
```




```
Out[11]= - Graphics -
```

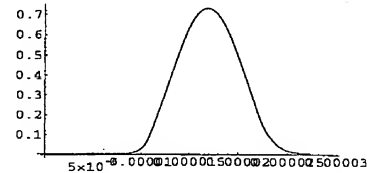
```
In[12]:= T
```

```
Out[12]= 3/812500
```

The unit of time is T = 1 for OptPulse. We scale the Pulses to $T = \frac{3}{812500}$ for the unit of time.

```
In[13]:= OptPulseScaled[8][0][t_] := OptPulse[L][0][t / T]
         OptPulseScaled[8][1][t_] := OptPulse[L][1][t / T]
```

```
In[14]:= Plot[OptPulseScaled[L][0][t], {t, 0, 8 T}]
```



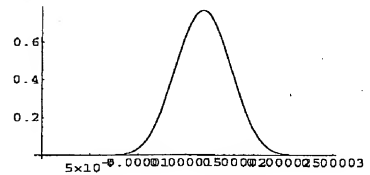

```
Out[14]= - Graphics -
```

```
In[16]:=
```

```
In[15]:= RandomBitSeq
```

```
Out[15]= {1, 1, -1, -1, -1, 1, 1, -1, 1, -1, 1, 1, -1, 1, -1, -1, -1, -1, 1, 1, 1, -1, -1, -1, 1, 1,
          1, -1, 1, -1, -1, 1, -1, -1, -1, 1, 1, 1, 1, -1, 1, -1, 1, -1, 1, 1, -1, 1, -1, -1, 1, 1, 1, 1, -1,
          1, -1, 1, -1, 1, -1, 1, 1, -1, -1, 1, -1, 1, -1, 1, 1, 1, 1, -1, -1, -1, 1, -1,
          1, 1, 1, -1, 1, -1, -1, 1, 1, 1, -1, -1, 1, 1, 1, 1, 1, 1, 1, 1, -1, -1, 1, -1, 1, -1}
```

```
In[16]:= Plot[FiltPulse[L][0][t], {t, 0, 8 T}]
```




```
Out[16]= - Graphics -
```

■ The Gold Sequence Set

We generate the sequences using the method specified by Serdar Boztas and P Vijay Kumar in Ref [1]. The numbering of the sequences is the one used in the paper. We generate a small subset of the sequences. There are $2^{10} + 1$ sequences with the quaternary polynomial used. Given any binary primitive polynomial, we can generate the corresponding quaternary polynomial.

```
In[22]:= Goldseqlist =
           SpecifiedGoldSequences[{1, 3, 2, 1, 0, 3, 0, 0, 1}][{1, 2, 3, Last - 1, Last}];
```

The last sequence of seqlist has nice autocorelation properties.

The last sequence is in fact a m−sequence of length 1023 bits.

```
In[23]:= Goldseqlist // Last

Out[23]= {1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0,
          0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0,
          1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0,
          1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0,
          0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0,
          0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1,
          1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
          1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0,
          1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0}

In[24]:= AutocorrelationSequence[Goldseqlist // Last]

Out[24]= {255, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
          -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
          -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
          -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
          -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
          -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
          -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
          -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
          -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
          -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
          -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
          -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1}
```

The third sequence in the list has the following autocorrelation

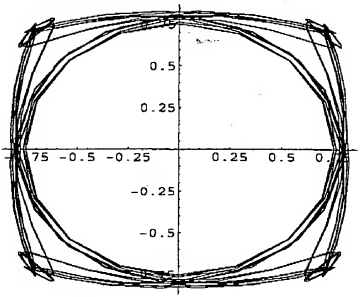
```
In[25]:= AutocorrelationSequence[Goldseqlist // #[[3]]&]

Out[25]= {255, -1, -1, -1, -1, -1, -1, -17, -1, -1, -1, -17, -1, -17, -17, -17, -1, -17, -1, -1,
          -1, -1, 15, -1, -1, 15, 15, -17, -17, -1, -17, 15, -1, -1, -17, 15, -1, 15, -1, 15, -1,
          15, -1, -1, 15, -17, -1, 15, -1, -1, -17, -17, -17, -1, -17, 15, 15, 15, -1, -1, 15,
          -17, 15, -1, -1, -1, -1, 15, 15, -1, 15, -1, -1, -17, -17, 15, -1, -1, 15, 15, -1, -1,
          -17, -1, -1, -17, -1, -1, 15, -1, 15, -17, -1, -1, -17, -1, -1, 15, -1, -17, 15, -1,
          15, -1, 15, 15, -1, 15, -17, 15, 15, -1, 15, -1, 15, -17, -1, -1, -1, 15, -17, -17, 15,
          -1, -17, -1, -1, -1, -1, -1, -1, -17, -1, 15, -17, -17, 15, -1, -1, -1, -17, 15, -1,
          15, -1, 15, 15, -17, 15, -1, 15, 15, -1, 15, -1, 15, -17, -1, 15, -1, -1, -17, -1,
          -1, -17, 15, -1, 15, -1, -1, -17, -1, -1, -17, -1, -1, 15, 15, -1, -1, 15,
          -17, -17, -1, -1, 15, -1, 15, 15, -1, -1, -1, -1, 15, -17, 15, -1, -1, 15,
          15, 15, -17, -1, -17, -17, -17, -1, -1, 15, -1, -17, 15, -1, -1, 15, -1, 15,
          -1, 15, -1, 15, -17, -1, -1, 15, -17, -1, -17, -17, 15, 15, -1, -1, 15, -1,
          -1, -1, -1, -17, -1, -17, -17, -17, -1, -17, -1, -1, -1, -17, -1, -1, -1, -1, -1, -1}
```

We generate the output of the modulator

```
In[93]:= ModOutput = Modulator[L][(Goldseqlist // Last) /. {0 -> -1}, NumberOfCurves → 2,
           ModulatingPulse → FiltPulse, SamplingInterval → T / 4];
```

We check the output

```
In[27]:= ListPlot[{Re[ModOutput], Im[ModOutput]} // Transpose, PlotJoined -> True,
          AspectRatio -> 1]
```




```
Out[27]= - Graphics -
```

Now we try to test the modulated sequence using the

The number of samples per chip is equal to

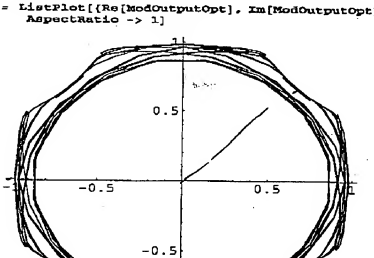
$$\frac{\frac{3}{26000000}}{T}$$

$$\frac{1}{32}$$

```
In[94]:= (Receiver[L][ModOutput, StartingQuadrant -> 0,
            SamplingInterval → T / 4, ModulatingPulse → FiltPulse] // Drop[#, 4]&) -
          ( (Goldseqlist // Last // Drop[#, -4]&) /. {0 -> -1})
```

```
Out[94]= {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          
```

```
In[96]:= ListPlot[{Re[ModOutputOpt], Im[ModOutputOpt]} // Transpose, PlotJoined -> True,
          AspectRatio -> 1]
```



```
Out[96]= - Graphics -

In[97]:= (Receiver[L][ModOutputOpt, StartingQuadrant -> 0,
             SamplingInterval → T / 4, ModulatingPulse → OptPulseScaled] // Drop[#, 4]& ) -
          ( (Goldseqlist // Last // Drop[#, -4]&) /. { 0 -> -1})

Out[97]= {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
```

■ CDMA OPERATION

As an example given a symbol stream e.g. {1,1,−1,−1, ...} consisting of −1,and 1, the following function demonstrates the encoding process.

```
In[49]:= CDMAEncode[BiPolarBitSeq_, GoldSeq_] :=
          Module[{x1, x2, x3},
          x1 = GoldSeq /. {0 → -1};
          Map[x1 # &, BiPolarBitSeq] // Flatten]

In[34]:= CDMAEncodedSeq = CDMAEncode[{-1, 1, 1, -1}, Goldseqlist // Last];
```

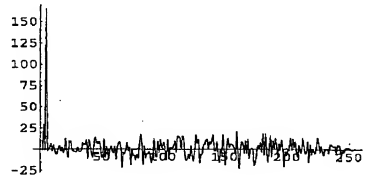
▪ CDMA decoding of single Symbol

The modular output associated with {1}

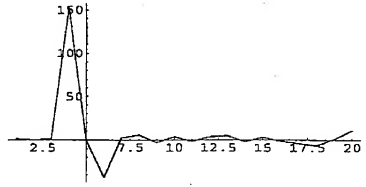
```
In[50]:= ModOutputPlusOne = Modulator[L][CDMAEncode[{1}, (Goldseqlist // Last) /. {0 -> -1}],
          NumberOfCurves → 2, ModulatingPulse → OptPulseScaled, SamplingInterval → T / 4];
```

This is a primitive decoder built to study the autocorrelation. This will help in decoding

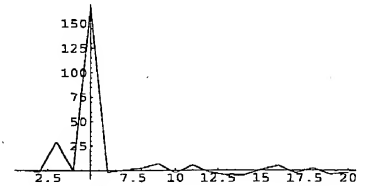
```
In[51]:= PrimitiveCDMAReceiver[ModOutput_, GoldSeq_, Sample_, OverSampling_] :=
          Module[{x1, x2State, x3Update, x4, x5State, x6},
           x1 = Partition[ModOutput, OverSampling] // Transpose // #[[Sample]]&;
           x2State = GoldSeq /. {0 -> -1};
            x5State = Table[(-1)^Mod[i, 2], {i, 0, Length[x2State] - 1}];
           x3Update := Module[{},
             x2State = RotateRight[x2State]; x4 = FoldList[Plus, 0, x2State] // Rest // I^#&;
            x5State = RotateRight[x5State]; x1 (x5State x4) // Apply[Plus, #]&];
          Table[x3Update, {i, 1, Length[GoldSeq]}]]

In[52]:= Tom = PrimitiveCDMAReceiver[ModOutputPlusOne, (Goldseqlist // Last), 1, 4];

In[53]:= ListPlot[Tom // Im, PlotJoined -> True, PlotRange -> All]
```




```
Out[53]= - Graphics -

In[54]:= ListPlot[Tom // Re // Take[#, 20]&, PlotJoined -> True, PlotRange -> All]
```

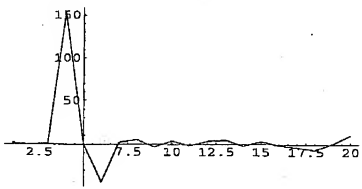



```
Out[54]= - Graphics -

In[55]:= ListPlot[Tom // Im // Take[#, 20]&, PlotJoined -> True, PlotRange -> All]
```

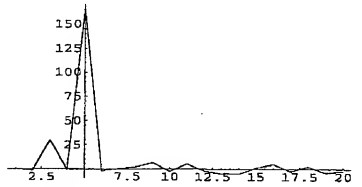



```
Out[55]= - Graphics -
```

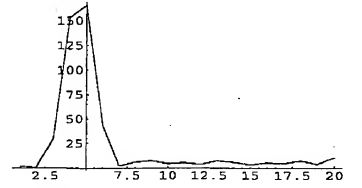
```
In[56]:= Take[Tom, 10]

Out[56]= {1.75501 + 0.329995 I, 0.0343421 - 1.58366 I, 1.39349 + 29.6902 I, 153.017 - 0.957068 I,
  -0.718377 + 165.563 I, -43.6424 - 2.04777 I, 2.21031 + 0.256636 I, 5.392 + 2.10642 I,
  -3.13017 + 6.54167 I, 3.61193 - 2.5452 I}

In[57]:= ListPlot[Tom // Re // Take[#, 20]&, PlotJoined -> True, PlotRange -> All]
```




```
Out[57]= - Graphics -

In[58]:= ListPlot[Tom // Im // Take[#, 20]&, PlotJoined -> True, PlotRange -> All]
```




```
Out[58]= - Graphics -

In[59]:= ListPlot[Tom // Abs // Take[#, 20]&, PlotJoined -> True, PlotRange -> All]
```

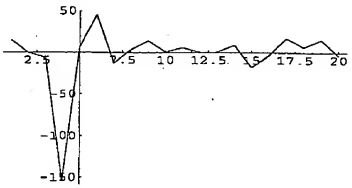



```
Out[59]= - Graphics -
```

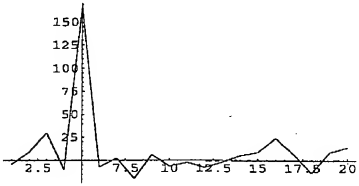
We try a less favourable sequence

```
In[60]:= ModOutputPlusOne3 = Modulator[L][CDMAEncode[{1}, Goldseqlist // #[[3]]&],
           NumberOfCurves → 2, ModulatingPulse → OptPulseScaled, SamplingInterval → T / 4];
```

```
In[68]:= Tom4 = PrimitiveCDMAReceiver[ModOutputPlusOne3, (Goldseqlist // #[[3]]&), 1, 4];

In[69]:= Take[Tom4, 10]

Out[69]= {14.7675 - 4.24542 I, -0.301874 + 8.61808 I, -5.92939 + 29.6038 I, -154.61 - 10.4754 I,
  6.64138 + 166.152 I, 45.2887 - 7.16233 I, -12.809 + 2.47796 I, 3.45948 - 19.7653 I,
  14.2097 + 6.33624 I, 0.585872 - 6.10244 I}

In[70]:= ListPlot[Tom4 // Re // Take[#, 20]&, PlotJoined -> True, PlotRange -> All]
```




```
Out[70]= - Graphics -

In[71]:= ListPlot[Tom4 // Im // Take[#, 20]&, PlotJoined -> True, PlotRange -> All]
```




```
Out[71]= - Graphics -
```

■ CDMA Decoding of Several Symbols with Training sequence Receiver

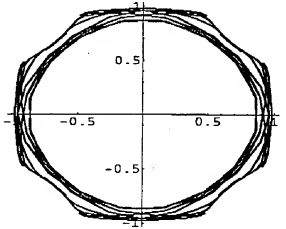
First we generate the modulator output. For simplicity we will use a very short training sequence . Let the training sequence be {1 ,−1,−1,1}. Let the guard sequences be {1,1,1} . Let the data symbols be generated by a random

```
In[72]:= data1 = {1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0} ;

In[73]:= data2 = {0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1};

In[74]:= guard = {1, 1, 1}

Out[74]= {1, 1, 1}

In[75]:= training = {0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1};
```

The GSM Training sequence is [0,0,1,0,0,1,0,1,1,1,0,0,0,0,1,0,0,0,1,0,0,1,0,,1,1,1]. In fact any short m−sequence can be used to characterise the output.

nly at this point that we differentially encode using the gsm scheme

```
In[76]:= frame = Join[guard, data1, training, data2, guard]

         General::spell1 :
          Possible spelling error: new symbol name "frame" is similar to existing symbol "Frame".

Out[76]= {1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0,
          0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1,
          1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1}

In[81]:= GSMDiffEncodedFrame[frame_] := Module[{x1, x2, x3}, x1 = Partition[frame, 2, 1];
           x2 = Map[Mod[#[[1]] + #[[2]], 2]&, x1] // Join[{frame[[1]]}, #]&;
          x3 = x2 /. {0 -> 1, 1 -> -1}]

In[83]:= frameEncoded = GSMDiffEncodedFrame[frame]

Out[83]= {-1, 1, 1, 1, 1, 1, 1, 1, 1, -1, -1, -1, -1, -1, -1, 1, 1, -1, -1, 1, -1, -1, -1, 1, 1,
          -1, -1, 1, -1, -1, -1, -1, 1, 1, 1, -1, 1, 1, 1, -1, 1, 1, 1, -1, -1, 1, -1, -1, -1,
          1, 1, -1, 1, -1, -1, -1, 1, -1, -1, -1, -1, 1, 1, -1, 1, -1, 1, -1, -1, 1, 1, 1, 1}

In[84]:= ModOutputFrame = Modulator[L][CDMAEncode[frameEncoded, Goldseqlist // Last],
           NumberOfCurves → 2, ModulatingPulse → OptPulseScaled, SamplingInterval → T/4];

In[85]:= Save["ModOutputFrameEncodedGSMLike.m", ModOutputFrame]

         << ModOutputFrame.m;

In[86]:= Length[ModOutputFrame]

Out[86]= 73440

In[87]:= TestData = Take[ModOutputFrame, {50, 7300}];

         General::spell1 : Possible spelling error: new symbol name "TestData" is similar
           to existing symbol "TextData".

In[88]:= TestData // (Re[#], Im[#])& // Transpose //
          ListPlot[#, PlotJoined -> True, PlotRange -> All, AspectRatio -> 1]&;
```



In the PrimitiveCDMA receiver we need to specify the sample. In the CDMA synchroniser we discover the sample.

```
In[89]:= CDMACoarseSynchroniser[ModOutput_, GoldSeq_, Threshold_, OverSampling_] :=
          Module[{x1, x2, x3, x4, x5State,
            x6Count, x6MaxCorr, seq, x7Update, x8, x9, x10, x11, x12, x13},
           x1 = ModOutput;
           x2 = GoldSeq /. 0 -> -1;
           x3 = Table[(-1)^Mod[i, 2], {i, 0, Length[x2] - 1}];
           x4 = FoldList[Plus, 0, x2] // Rest // I^#& // x3 # &;
           x5State = Take[x1, (Length[x2] + 1) OverSampling];
           x6Count = 1;
           x6MaxCorr = 0;
           seq = Drop[x1, OverSampling (Length[x2] + 1)];
          x7Update := Module[{},
         x8 = Partition[x5State, OverSampling] // Transpose;
         x9 = Map[{Drop[#, -1] . x4, Drop[#, 1] - x4}&, x8];
         x10 = Map[{ Abs[Im[#[[1]]]],
                Abs[Re[#[[1]]]] , Abs[Re[#[[2]]]] , Abs[Im[#[[2]]] ]&, x9];
         x11 = If[Max[x10] > Threshold, Throw[{x10, x6Count, True}],
                {x6Count, x10, x6MaxCorr = Max[x6MaxCorr, x10], False}];
             x6Count = x6Count + 1;
             x5State = Join[Drop[x5State, OverSampling], Take[seq , OverSampling]];
             seq = Drop[seq, OverSampling];
             x11 ];
         x12 = Catch[Table[x7Update, {i, 1, Length[seq] / OverSampling }]];
          x13 = If[Last[x12] === True,
            CDMAFineSynchroniser[ModOutput, x12[[2]], x4, OverSampling],
            {"Failed to Coarse Synchronise", False}]]

In[90]:= CDMAFineSynchroniser[ModOutput_,
           ThresholdCorrelatioCount_, CorrelatingSeq_, OverSampling_] :=
            Module[{x1, x2, x3, x4, x5, x6, seq, x7Update, x8First,
             x8Second, x9First, x9Second, x10First, x10Second, x11, x12, x13, x14},
            x1 = If[ThresholdCorrelatioCount > 3,
              ThresholdCorrelatioCount - 3, ThresholdCorrelatioCount];
            x2 = CorrelatingSeq;
            x3 = Drop[ModOutput, x1 OverSampling];
           x4 = CDMAPositionFinder[x3, x2, OverSampling];
           x5 = CDMAPositionFinder[Drop[x3, Length[x2] OverSampling] , x2, OverSampling];
            x6 = CDMAPositionFinder[Drop[x3, 2 Length[x2] OverSampling] , x2, OverSampling];
           x7 = PositionAverager[{x4[[1]], x5[[1]], x6[[1]]}];
           x8First = Drop[x3, (x7[[1]] - 1) OverSampling] //
                Partition[#, OverSampling]& // Transpose // #[[x7[[2]]]]&;
            x8Second = Drop[x3, x7[[1]] OverSampling] //
                Partition[#, OverSampling]& // Transpose // #[[x7[[2]]]]&;
          x9First = x8First // Partition[#, Length[x2]]&;
           x10First = Map[Function[x, x . x2], x9First];
             x9Second = x8Second // Partition[#, Length[x2]]&;
           x10Second = Map[Function[x, x . x2], x9Second] ;
           DeModulator[{x10First, x10Second}]  ]

In[91]:= CDMAPositionFinder[ModOutput_, CorrelatingSeq_, OverSampling_] :=
           Module[{x5State, x6Count, seq, x7Update, x8, x9, x10, x11, x12, x13},
          x5State = Take[ModOutput, (Length[CorrelatingSeq] + 1) OverSampling];
           x6Count = 1;
           seq = Drop[ModOutput, OverSampling (Length[CorrelatingSeq] + 1)];
            x7Update := Module[{},
         x8 = Partition[x5State, OverSampling] // Transpose;
         x9 = Map[{Drop[#, -1] . CorrelatingSeq, Drop[#, 1] . CorrelatingSeq}&, x8];
             x6Count = x6Count + 1;
             x5State = Join[Drop[x5State, OverSampling], Take[seq , OverSampling]];
             seq = Drop[seq, OverSampling];
             x9 ];
         x10 = Table[x7Update, {i, 1, 10}];
         x11 = MapIndexed[Max[{Abs[Re[#]], Abs[Im[#]]}]&, x10, {3}];
         x12 = MapIndexed[Apply[Plus, #]&, x11, {2}];
           x13 = Position[x12, Max[x12]] ]
```

We need to define a position averager. We for now just take the first element

```
In[98]:= PositionAverager[PositionList_] := First[PositionList]
```

```
In[101]:= Tom4 =
           CDMACoarseSynchroniser[TestData // Drop[#, 250 4]&, Goldseqlist // Last , 100, 4]

Out[101]= DeModulator[{{-157.056 - 0.691379 I, -0.691379 - 157.056 I,
            157.056 - 0.691379 I, 0.691379 + 157.056 I, -157.056 + 0.691379 I},
           {-169.734 - 0.510132 I, 0.510132 - 169.734 I, 169.734 + 0.510132 I,
            -0.510132 + 169.734 I, -169.734 - 0.510132 I}}]

In[102]:= Tom5 = CDMACoarseSynchroniser[ModOutputFrame // Drop[#, 250 4]&,
             Goldseqlist // Last , 100, 4]

Out[102]= DeModulator[{{1.02745 + 156.682 I, -157.056 + 0.691379 I, -0.691379 - 157.056 I,
            157.056 - 0.691379 I, 0.691379 + 157.056 I, -157.056 + 0.691379 I,
            -0.691379 - 157.056 I, 157.25 - 0.929033 I, 19.1272 - 6.93509 I, -7.37139 - 18.8305 I,
            -18.8305 + 7.37139 I, 7.37139 + 18.8305 I, 18.8305 - 7.37139 I, -7.57128 - 19.0518 I,
            -1.02745 - 156.682 I, 157.25 - 0.929033 I, 19.1272 - 6.93509 I, -7.57128 - 19.0518 I,
            -1.2651 - 156.875 I, -6.93509 - 19.1272 I, -18.8305 + 7.37139 I, 7.57128 + 19.0518 I,
            1.02745 + 156.682 I, -157.25 + 0.929033 I, -19.1272 + 6.93509 I, 7.57128 + 19.0518 I,
            1.2651 + 156.875 I, 6.93509 + 19.1272 I, 18.8305 - 7.37139 I, -7.57128 - 19.0518 I,
            -1.02745 - 156.682 I, 157.25 - 0.929033 I, 19.3484 - 7.13498 I, 156.875 + 1.2651 I,
            1.02745 + 156.682 I, -157.25 + 0.929033 I, -19.1272 + 6.93509 I, 7.57128 + 19.0518 I,
            0.691379 + 157.056 I, -157.25 + 0.929033 I, -19.1272 + 6.93509 I, 7.57128 + 19.0518 I,
            1.02745 + 156.682 I, -157.25 + 0.929033 I, -19.1272 + 6.93509 I, 7.57128 + 19.0518 I,
            1.2651 + 156.875 I, 6.93509 + 19.1272 I, 18.8305 - 7.37139 I, -7.57128 - 19.0518 I,
            -1.02745 - 156.682 I, 157.25 - 0.929033 I, 19.3484 - 7.13498 I, 156.875 - 1.2651 I,
            19.1272 - 6.93509 I, -7.37139 - 18.8305 I, -19.0518 + 7.57128 I, -156.875 + 1.2651 I,
            -19.1272 + 6.93509 I, 7.37139 + 18.8305 I, 18.8305 - 7.37139 I, -7.57128 - 19.0518 I,
            -1.2651 - 156.875 I, -6.93509 - 19.1272 I, -19.0518 + 7.57128 I,
            -156.875 + 1.2651 I, -19.1272 + 6.93509 I, 7.37139 + 18.8305 I,
            19.0518 - 7.57128 I, 156.875 - 1.2651 I, 19.3484 - 7.13498 I, 156.682 - 1.02745 I,
            0.691379 + 157.056 I, -157.056 + 0.691379 I}, {-0.478141 + 169.715 I,
            -169.734 - 0.510132 I, 0.510132 - 169.734 I, 169.734 + 0.510132 I,
            -0.510132 + 169.734 I, -169.734 - 0.510132 I, 0.510132 - 169.734 I,
            169.813 - 0.714545 I, 20.5326 - 6.8667 I, -6.8871 - 20.5026 I, -20.5026 + 6.8871 I,
            6.8871 + 20.5026 I, 20.5026 - 6.8871 I, -7.42576 - 21.3328 I, 0.478141 - 169.715 I,
            169.813 - 0.714545 I, 20.5326 - 6.8667 I, -7.42576 - 21.3328 I, -0.746535 - 169.794 I,
            -6.8667 - 20.5326 I, -20.5026 + 6.8871 I, 7.42576 + 21.3328 I, -0.478141 + 169.715 I,
            -169.813 + 0.714545 I, -20.5326 + 6.8667 I, 7.42576 + 21.3328 I, 0.746535 + 169.794 I,
            6.8667 + 20.5326 I, 20.5026 - 6.8871 I, -7.42576 - 21.3328 I, 0.478141 - 169.715 I,
            169.813 - 0.714545 I, 21.3629 - 7.40536 I, 169.715 + 0.478141 I,
            -0.510132 + 169.734 I, -169.813 + 0.714545 I, -20.5326 + 6.8667 I,
            7.42576 + 21.3328 I, -0.478141 + 169.715 I, -169.813 + 0.714545 I,
            -20.5326 + 6.8667 I, 7.42576 + 21.3328 I, 0.746535 + 169.794 I, 6.8667 + 20.5326 I,
            20.5026 - 6.8871 I, -7.42576 - 21.3328 I, 0.478141 - 169.715 I, 169.813 - 0.714545 I,
            21.3629 - 7.40536 I, 169.794 - 0.746535 I, 20.5326 - 6.8667 I, -6.8871 - 20.5026 I,
            -21.3328 + 7.42576 I, -169.794 + 0.746535 I, -20.5326 + 6.8667 I, 6.8871 + 20.5026 I,
            20.5026 - 6.8871 I, -7.42576 - 21.3328 I, -0.746535 - 169.794 I, -6.8667 - 20.5326 I,
            -21.3328 + 7.42576 I, -169.794 + 0.746535 I, -20.5326 + 6.8667 I,
            6.8871 + 20.5026 I, 21.3328 - 7.42576 I, 169.794 - 0.746535 I, 21.3629 - 7.40536 I,
            169.715 + 0.478141 I, -0.510132 + 169.734 I, -169.734 - 0.510132 I}}]

In[105]:= SeqI = Table[I^(i) (-1)^(i + 1), {i, 1, Length[Tom5[[1, 1]]]}]

Out[105]= {I, 1, -I, -1, I, 1, -I, -1, I, 1, -I, -1, I, 1, -I, -1, I, 1, -I, -1, I, 1, -I,
            -1, I, 1, -I, -1, I, 1, -I, -1, I, 1, -I, -1, I, 1, -I, -1, I, 1, -I, -1, I,
            1, -I, -1, I, 1, -I, -1, I, 1, -I, -1, I, 1, -I, -1, I, 1, -I, -1, I, 1}
```

```
In[106]:= Tom5[[1, 1]] SeqI

Out[106]= {-156.682 + 1.02745 I, -157.056 + 0.691379 I, -157.056 + 0.691379 I,
  -157.056 + 0.691379 I, -157.056 + 0.691379 I, -157.056 + 0.691379 I,
  -157.056 + 0.691379 I, -157.25 + 0.929033 I, 6.93509 + 19.1272 I, -7.37139 - 18.8305 I,
  7.37139 + 18.8305 I, -7.37139 - 18.8305 I, 7.37139 + 18.8305 I, -7.57128 - 19.0518 I,
  -156.682 + 1.02745 I, -157.25 + 0.929033 I, 6.93509 + 19.1272 I, -7.57128 - 19.0518 I,
  -156.875 + 1.2651 I, 6.93509 + 19.1272 I, -7.37139 - 18.8305 I, 7.57128 + 19.0518 I,
  156.682 - 1.02745 I, 157.25 - 0.929033 I, -6.93509 - 19.1272 I, 7.57128 + 19.0518 I,
  156.875 - 1.2651 I, -6.93509 - 19.1272 I, 7.37139 + 18.8305 I, -7.57128 - 19.0518 I,
  -156.682 + 1.02745 I, -157.25 + 0.929033 I, 7.13498 + 19.3464 I, 156.682 - 1.02745 I,
  157.056 - 0.691379 I, 157.25 - 0.929033 I, -6.93509 - 19.1272 I, 7.57128 + 19.0518 I,
  156.682 - 1.02745 I, 157.25 - 0.929033 I, -6.93509 - 19.1272 I, 7.57128 + 19.0518 I,
  156.875 - 1.2651 I, -6.93509 - 19.1272 I, 7.37139 + 18.8305 I, -7.57128 - 19.0518 I,
  -156.682 + 1.02745 I, -157.25 + 0.929033 I, 7.13498 + 19.3464 I, 156.875 - 1.2651 I,
  -6.93509 - 19.1272 I, 7.37139 + 18.8305 I, -7.57128 - 19.0518 I, -156.875 + 1.2651 I,
  6.93509 + 19.1272 I, -7.37139 - 18.8305 I, 7.37139 + 18.8305 I, -7.57128 - 19.0518 I,
  -156.875 + 1.2651 I, 6.93509 + 19.1272 I, -7.57128 - 19.0518 I, -156.875 + 1.2651 I,
  6.93509 + 19.1272 I, -7.37139 - 18.8305 I, 7.57128 + 19.0518 I, 156.875 - 1.2651 I,
  -7.13498 - 19.3464 I, -156.682 + 1.02745 I, -157.056 + 0.691379 I,
  -157.056 + 0.691379 I}

In[114]:= demodframe = Sign[Re[Tom5[[1, 1]] SeqI]] /. {-1 -> 1, 1 -> 0}

Out[114]= {1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0,
  0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0,
  1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1}
```

The first and last bits have been lost in the processing

```
In[109]:= frame

Out[109]= {1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1,
  0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1,
  0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1}

In[122]:= Length[frame]

Out[122]= 72

In[115]:= truncatedframe = frame // Drop[#, 1]& // Drop[#, -1]&

Out[115]= {1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0,
  0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0,
  1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1}

In[116]:= demodframe - truncatedframe

Out[116]= {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
```

■ References

1 "Binary Sequences with Gold-Like Correlation but Larger Linear Span" by Serdar Boztas and P. Vijay Kumar IEEE Trans. on Information Theory Vol 40 No 2, March 1994

```
BeginPackage["RingFunctionsDiffEncoded`"]

RingDivision::usage = "RingDivision[ModuloRing][{Denominator,Numerator}] =
   {Quotitent,Remainder,Denominator,Flag} e.g If num = {1,0,0,0,0,0,0,0,0,1,
   0,0,0,0,0,1},den = {1,0,0,1,0,1}, then RingDivision[2][{den,num}] = {{1,
   0,0,1,0,1,1,0,0,0,1},{1,0,1,0,0},{1,0,0,1,0,1},True}. When the
   algorithm encounters a zero devisor the flag is set to False. e.g RingDivision[
   4][{{2,1,1,1},{1,0,0,1}}] = {{},{1,0,0,1},{2,1,1,1},False} "

DropLeadingZeros::usage =
  "DropLeadingZeros[DenominatorSeq] drops the leading zeros in the sequence.
     e.g DropLeadingZeros[{3,3,1}] = {3,3,1}.  DropLeadingZeros[{0,0,0}] = {}"

PossibleDivisors::usage = "PossibleDivisors[ModuloRing][Order] gives the list of
   possible devisors. Note the MS power of the poly is LHS. PossibleDivisors[4][
   2] = {{1},{2},{3},{1,0},{1,1},{1,2},{1,3},{2,0},{2,1},{2,2},{2,3},{3,0},
   {3,1},{3,2},{3,3},{1,0,0},{1,0,1},{1,0,2},{1,0,3},{1,1,0},{1,1,1},{1,
   1,2},{1,1,3},{1,2,0},{1,2,1},{1,2,2},{1,2,3},{1,3,0},{1,3,1},{1,3,2},
   {1,3,3},{2,0,0},{2,0,1},{2,0,2},{2,0,3},{2,1,0},{2,1,1},{2,1,2},{2,1,
   3},{2,2,0},{2,2,1},{2,2,2},{2,2,3},{2,3,0},{2,3,1},{2,3,2},{2,3,3},
   {3,0,0},{3,0,1},{3,0,2},{3,0,3},{3,1,0},{3,1,1},{3,1,2},{3,1,
   3},{3,2,0},{3,2,1},{3,2,2},{3,2,3},{3,3,0},{3,3,1},{3,3,2},{3,3,3}}"

PolynomialMultiplication::usage = "PolynomialMultiplication[ModuloRing][
   poly1,poly2] performs polynomial multiplication moduloRing e.g.
   PolynomialMultiplication[4][{2,1},{2,1}] = {1}"

ModuloMultplication::usage =
  "ModuloMultplication[ModuloRing][PrimitivePolynomial][poly1,poly2] e.
     g ModuloMultplication[4][{1,2,1,3}][{3,3,0},{3,3,0}] = {1,0}"

RingPower::usage = "RingPower[RingModulo][PrimitiveElement][Element] e.g RingPower[
   4][{1,0,0,3,2,3}][{1,0,0}] = {{1,0,0},{1,0,0,0,0},{1,2,1,0},{2,1,1,
   2,1},{1,0,2,0,1},{2,1,3,1,0},{3,3,1,0,1},{1,3,2,1,3},{2,2,0,3,
   3},{1,1,2,2},{1,2,3,2,1},{3,3,1,1,2},{1,0,3,1,3},{3,2,1,1,
   0},{1,0,0,3,2},{1,0},{1,0,0,0},{1,2,1},{1,2,1,0,0},{1,1,0,
   1,2},{2,1,3,1},{1,3,3,0,2},{3,1,3,3,3},{3,2,2,1,1},{2,0,1,
   3,2},{1,1,2,2,0},{2,3,3,3,1},{3,1,0,0,3},{3,2,1,1},{2,1,0,2,3},{1}}"

CyclicMultiplativeGroup::usage = "CyclicMultiplativeGroup[ModuloRing][
   PrimitivePolynomial] e.g. CyclicMultiplativeGroup[4][{1,2,1,3}] = {{1},{1,
   0},{1,0,0},{2,3,1},{3,3,2},{1,3,3},{1,2,1}} and  CyclicMultiplativeGroup[
   4][{1,0,0,3,2,3}] = {{1},{1,0,0},{1,0,0,0,0},{1,2,1,0},{1,2,1,1,2,1},
   {1,0,2,0,1},{2,1,3,1,0},{3,3,1,0,1},{1,3,2,1,3},{2,2,0,3,3},{1,1,2,
   2},{1,2,3,2,1},{3,3,1,1,2},{1,0,3,1,3},{3,2,1,1,0},{1,0,0,3,2},
   {1,0},{1,0,0,0},{1,2,1},{1,2,1,0,0},{1,1,0,1,2},{2,1,3,1},
   {1,3,3,0,2},{3,1,3,3,3},{3,2,2,1,1},{2,0,1,3,2},{1,1,2,2,
   0},{2,3,3,3,1},{3,1,0,0,3},{3,2,1,1},{2,1,0,2,3}}"

ZeroSequences::usage =
  "ZeroSequences[ModuloRing][Order] generates the zero ideal ZeroSequences[4][
     3]= {{0,0,0},{0,0,2},{0,2,0},{0,2,2},{2,0,0},{2,0,2},{2,2,0},{2,2,2}}"

ZeroPad::usage = "ZeroPad[Order][Element] eg ZeroPad[3][{1}] = {0,0,1}"
```

```
TwoAdicExpansion::usage = "TwoAdicExpansion[ModuloRing][PrimitivePolynomial] e.g
    TwoAdicExpansion[4][{1,2,1,3}] = {{{0,0,0},{0,0,0}},{{0,0,0},{0,0,1},
    {2}},{{0,0,0},{0,1,0},{2,0}},{{0,0,0},{1,0,0},{2,0,0}},{{0,0,0},{2,3,
    1},{2,2}},{{0,0,0},{3,3,2},{2,2,0}},{{0,0,0},{1,3,3},{2,2,2}},{{0,0,0},
    {1,2,1},{2,0,2}},{{0,0,1},{0,0,0},{1}},{{0,0,1},{0,0,1},{3}},{{0,0,1},
    {0,1,0},{3,1}},{{0,0,1},{1,0,0},{2,0,1}},{{0,0,1},{2,3,1},{2,3}},{{0,0,
    1},{3,3,2},{2,2,1}},{{0,0,1},{1,3,3},{2,2,3}},{{0,0,1},{1,2,1},{2,0,
    3}},{{0,1,0},{0,0,0},{1,0}},{{0,1,0},{0,0,1},{1,2}},{{0,1,0},{0,1,0},
    {3,0}},{{0,1,0},{1,0,0},{2,1,0}},{{0,1,0},{2,3,1},{3,2}},{{0,1,0},{3,3,
    2},{2,3,0}},{{0,1,0},{1,3,3},{2,3,2}},{{0,1,0},{1,2,1},{2,1,2}},{{1,0,
    0},{0,0,0},{1,0,0}},{{1,0,0},{0,0,1},{1,0,2}},{{1,0,0},{0,1,0},{1,2,
    0}},{{1,0,0},{1,0,0},{3,0,0}},{{1,0,0},{2,3,1},{1,2,2}},{{1,0,0},{3,3,
    2},{3,2,0}},{{2,0,0},{1,3,3},{3,2,2}},{{1,0,0},{1,2,1},{3,0,2}},{{2,3,
    1},{0,0,0},{2,3,1}},{{2,3,1},{0,0,1},{2,3,3}},{{2,3,1},{0,1,0},{2,1,
    1}},{{2,3,1},{1,0,0},{3,1}},{{2,3,1},{2,3,1},{2,1,3}},{{2,3,1},{3,3,2},
    {1,1}},{{2,3,1},{1,3,3},{1,3}},{{2,3,1},{1,2,1},{3,3}},{{3,3,2},{0,0,
    0},{3,3,2}},{{3,3,2},{0,0,1},{3,3,0}},{{3,3,2},{0,1,0},{3,1,2}},{{3,3,
    2},{1,0,0},{1,3,2}},{{3,3,2},{2,3,1},{3,1,0}},{{3,3,2},{3,3,2},{1,1,
    2}},{{3,3,2},{1,3,3},{1,1,0}},{{3,3,2},{1,2,1},{1,3,0}},{{1,3,3},{0,0,
    0},{1,3,3}},{{1,3,3},{0,0,1},{1,3,1}},{{1,3,3},{0,1,0},{1,1,3}},{{1,3,
    3},{1,0,0},{3,3,3}},{{1,3,3},{2,3,1},{1,1,1}},{{1,3,3},{3,3,2},{3,1,
    3}},{{1,3,3},{1,3,3},{3,1,1}},{{1,3,3},{1,2,1},{3,3,1}},{{1,2,1},{0,0,
    0},{1,2,1}},{{1,2,1},{0,0,1},{1,2,3}},{{1,2,1},{0,1,0},{1,0,1}},{{1,
    2,1},{1,0,0},{3,2,1}},{{1,2,1},{2,3,1},{1,0,3}},{{1,2,1},{3,3,2},{3,0,1}},{{1,2,1},{1,3,3},{3,0,3}},{{1,2,1},{1,2,1},{3,2,3}})"

σ::usage = "σ[ModuloRing][PrimitivePolynomial][TwoadicElement] e.g "

AutomorphismSigma::usage = "AutomorphismSigma[ModuloRing][PrimitivePolynomial] e.g
    AutomorphismSigma[4][{1,2,1,3}] = {{{0,0,0},{0,0,0}},{{0,0,2},{0,0,2}},
    {{0,2,0},{2,0,0}},{{2,0,0},{2,2,0}},{{0,2,2},{2,0,2}},{{2,2,0},{0,2,
    0}},{{2,2,2},{0,2,2}},{{2,0,2},{2,2,2}},{{0,0,1},{0,0,1}},{{0,0,3},{0,
    0,3}},{{0,2,1},{2,0,1}},{{2,0,1},{2,2,1}},{{0,2,3},{2,0,3}},{{2,2,1},
    {0,2,1}},{{2,2,3},{0,2,3}},{{2,0,3},{2,2,3}},{{0,1,0},{1,0,0}},{{0,1,
    2},{1,0,2}},{{0,1,0},{3,0,0}},{{2,1,0},{3,2,0}},{{0,3,2},{3,0,2}},{{2,
    3,0},{1,2,0}},{{2,3,2},{1,2,2}},{{2,1,2},{3,2,2}},{{1,0,0},{3,3,2}},
    {{1,0,2},{3,3,0}},{{1,2,0},{1,3,2}},{{3,0,0},{1,1,2}},{{1,2,2},{1,3,
    0}},{{3,2,0},{3,1,2}},{{3,2,2},{3,1,0}},{{3,0,2},{1,1,0}},{{2,3,1},{1,
    2,1}},{{2,3,3},{1,2,3}},{{2,1,1},{3,2,1}},{{0,3,1},{3,0,1}},{{2,1,3},
    {3,2,3}},{{0,1,1},{1,0,1}},{{0,1,3},{1,0,3}},{{0,3,3},{3,0,3}},{{3,3,
    2},{0,1,0}},{{3,3,0},{0,1,2}},{{3,1,2},{2,1,0}},{{1,3,2},{2,
    3,0}},{{3,1,0},{2,1,2}},{{1,1,2},{0,3,0}},{{1,1,0},{0,3,2}},
    {{1,3,0},{2,3,2}},{{1,3,3},{2,3,1}},{{3,1,1},{2,3,3}},{{1,1,
    3},{0,3,1}},{{3,3,3},{0,1,1}},{{1,1,1},{0,3,3}},{{3,1,3},{2,
    1,1}},{{3,1,1},{2,1,3}},{{3,3,1},{0,1,3}},{{1,2,1},{1,3,3}},
    {{1,3,3},{1,3,1}},{{1,0,1},{3,3,3}},{{3,2,1},{3,1,3}},{{1,0,
    3},{3,3,1}},{{3,0,1},{1,1,3}},{{3,0,3},{1,1,1}},{{3,2,3},{3,1,1}})"

UnitsRing::usage = "UnitsRing[ModuloRing][PrimitivePolynomial] e.g. UnitsRing[4][
    {1,2,1,3}] = {{0,0,1},{0,0,3},{0,1,0},{0,1,1},{0,1,2},{0,1,3},{0,2,
    1},{0,2,3},{0,3,0},{0,3,1},{0,3,2},{0,3,3},{1,0,0},{1,0,1},{1,0,2},{1,
    0,3},{1,1,0},{1,1,1},{1,1,2},{1,1,3},{1,2,0},{1,2,1},{1,2,2},{1,2,3},
    {1,3,0},{1,3,1},{1,3,2},{1,3,3},{2,0,1},{2,0,3},{2,1,0},{2,1,1},{2,1,
    2},{2,1,3},{2,2,1},{2,2,3},{2,3,0},{2,3,1},{2,3,2},{2,3,3},{3,1,
    0},{3,0,1},{3,0,2},{3,0,3},{3,1,0},{3,1,1},{3,1,2},{3,1,
    3},{3,2,0},{3,2,1},{3,2,2},{3,2,3},{3,3,0},{3,3,1},{3,3,2},{3,3,3}})"

𝒯::usage = "𝒯[Order_][𝔬_][𝔵_] e.g 𝒯[4][x3][{3, 0, 3}] = "
```

```
TraceRepresentation::usage = "TraceRepresentation[ModuloRing_][
   PrimitivePolynomial_] e.g. TraceRepresentation[4][{1,2,1,3}] = {{{0,0,0}, {0,
   0,0}}, {{0,0,2}, {0,0,0}}, {{0,2,0}, {2,2,0}}, {{2,0,0}, {0,2,0}}, {{0,2,2},
   {2,2,0}}, {{2,2,0}, {2,0,0}}, {{2,2,2}, {2,0,0}}, {{2,0,2}, {0,2,0}}, {{0,0,
   1}, {0,0,2}}, {{0,0,3}, {0,0,2}}, {{0,2,1}, {2,2,2}}, {{2,0,1}, {0,2,2}}, {{0,
   2,3}, {2,2,2}}, {{2,2,1}, {2,0,2}}, {{2,2,3}, {2,0,2}}, {{2,0,3}, {0,2,2}},
   {{0,1,0}, {1,1,0}}, {{0,1,2}, {1,1,0}}, {{0,3,0}, {3,3,0}}, {{2,1,0}, {1,3,
   0}}, {{0,3,2}, {3,3,0}}, {{2,3,0}, {3,1,0}}, {{2,3,2}, {3,1,0}}, {{2,1,2}, {1,
   3,0}}, {{1,0,0}, {0,3,2}}, {{1,0,2}, {0,3,2}}, {{1,2,0}, {2,1,2}}, {{3,0,0},
   {0,1,2}}, {{1,2,2}, {2,1,2}}, {{3,2,0}, {2,3,2}}, {{3,2,2}, {2,3,2}}, {{3,0,
   2}, {0,1,2}}, {{2,3,1}, {3,1,2}}, {{2,3,3}, {3,1,2}}, {{2,1,1}, {1,3,2}}, {{0,
   3,1}, {3,3,2}}, {{2,1,3}, {1,3,2}}, {{0,1,1}, {1,1,2}}, {{0,1,3}, {1,1,2}},
   {{0,3,3}, {3,3,2}}, {{3,3,2}, {3,0,2}}, {{3,3,0}, {3,0,2}}, {{3,1,2}, {1,2,
   2}}, {{3,3,2}, {3,2,2}}, {{3,1,0}, {1,2,2}}, {{1,1,2}, {1,0,2}}, {{1,1,0}, {1,
   0,2}}, {{1,3,0}, {3,2,2}}, {{1,3,3}, {3,2,0}}, {{1,3,1}, {3,2,0}}, {{1,1,
   3}, {1,0,0}}, {{3,3,3}, {3,0,0}}, {{1,1,1}, {1,0,0}}, {{3,1,3}, {1,
   2,0}}, {{3,1,1}, {1,2,0}}, {{3,3,1}, {3,0,0}}, {{1,2,1}, {2,1,0}},
   {{1,2,3}, {2,1,0}}, {{1,0,1}, {0,3,0}}, {{3,2,1}, {2,3,0}}, {{1,0,
   3}, {0,3,0}}, {{3,0,1}, {0,1,0}}, {{3,0,3}, {0,1,0}}, {{3,2,3}, {2,3,0}}}"

TupleRepresentation::usage = "TupleRepresentation[ModuloRing][PrimitivePolynomial]
   designed for fields. e.g.TupleRepresentation[2][{1,0,0,1,0,1}] = {{0,0,0,
   0,1}, {0,0,0,1,0}, {0,0,1,0,0}, {0,1,0,0,0}, {1,0,0,0,0}, {0,0,1,0,1},
   {0,1,0,1,0}, {1,0,1,0,0}, {0,1,1,0,1}, {1,1,0,1,0}, {1,0,0,0,1}, {0,0,
   1,1,1}, {0,1,1,1,0}, {1,1,1,0,1}, {1,1,0,1,1}, {1,
   1,0,1,1}, {1,0,1,1,1}, {0,0,0,1,1}, {0,0,1,1,0}, {0,1,1,0,0},
   {1,1,0,0,0}, {1,0,1,0,1}, {0,1,1,1,1}, {1,1,1,1,0}, {1,1,0,0,
   1}, {1,0,1,1,1}, {0,1,0,1,1}, {1,0,1,1,0}, {0,1,0,0,1}, {1,0,0,1,
   0}"

MinimumPoly::usage = "MinimumPoly[ModuloRing][PrimitivePolynomial][rootPower][
   var]  designed for fields.e.g. MinimumPoly[2][{1,0,0,1,0,1}][x] "

GraeffeMethod::usage = "GraeffeMethod[GF2minpoly] generates a primitive
   polynomial in rung z4 e.g GraeffeMethod[{1,0,0,1,0,1}] = {1,0,0,3,2,3}  "

SequenceGenerator::usage =
  "SequenceGenerator[ModuloRing][PrimitivePoly][InitialCond][SeqLength]
    e.g. SequenceGenerator[4][{1, 0, 0, 3, 2, 3}][{0, 0, 0, 0, 2}][31] =
   {{2, 0, 0, 0, 0, 2, 0, 0, 2, 0, 2, 2, 0,
     0, 2, 2, 2, 2, 2, 2, 0, 0, 0, 2, 2, 0, 2, 2, 2, 0, 2, 0}, {0, 0, 0, 0, 2}}
   "

InitialConditions::usage = "InitialConditions[PrimitivePoly] e.g InitialConditions[
   {1,0,0,3,2,3}] = {{0,0,0,0,1},{0,0,0,0,3},{0,0,0,2,1},{0,0,0,2,
   3},{0,0,2,0,1},{0,0,2,0,3},{0,0,2,2,1},{0,0,2,2,3},{0,2,0,
   0,1},{0,2,0,0,3},{0,2,0,2,1},{0,2,0,2,3},{0,2,2,0,1},{0,2,
   2,0,3},{0,2,2,2,1},{0,2,2,2,3},{2,0,0,0,1},{2,0,0,0,3},{2,
   0,0,2,1},{2,0,0,2,3},{2,0,2,0,1},{2,0,2,0,3},{2,0,2,2,1},
   {2,0,2,2,3},{2,2,0,0,1},{2,2,0,0,3},{2,2,0,2,1},{2,2,0,2,
   3},{2,2,2,0,1},{2,2,2,0,3},{2,2,2,2,1},{2,2,2,2,3},{0,0,0,0,2}} "

GoldSequence::usage = "GoldSequence[PrimitivePoly][InitCondNumber] e.g.
   GoldSequence[{1,0,0,3,2,3}][33] = {1,0,0,0,0,1,0,0,1,0,
   1,0,0,1,1,1,1,1,1,0,0,0,1,1,0,1,1,1,0,1,0} "

AutocorrelationSequence::usage = "AutocorrelationSequence[GoldSequence] e.g.
   AutocorrelationSequence[{1,0,0,0,0,1,0,0,1,0,1,1,0,0,1,1,1,1,1,0,
   0,0,1,1,0,1,1,1,0,1,0}] = {31,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,
   -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1} "

SpecifiedGoldSequences::usage = "SpecifiedGoldSequences[PrimitivePoly_][
   NumberList_] e.g. SpecifiedGoldSequences[{1,0,0,3,2,3}][{1,2,3,Last -
   1,Last}] = {{0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,1,1,0,1,1,1,
   0,0,0,1,1,1,0,0,1,1,1},{1,0,0,0,0,1,0,0,1,1,1,1,0,1,1,0,
   0,1,0,1,1,1,1,0,0,1,0,1,1,0,0,0,0,1,1,1,1,0,1,0,1,0,0,0,
   1,1,1,0,0,0,1,0,0,1,0,0,1,0,1,1,1,1,0},{1,1,1,1,1,0,
   0,0,1,0,0,1,1,1,1,0,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
   0,0,0,1,0,0,1,1,1,1,1,1,0,1,0,1,1,1,1,1,1,1,1,1,1,1,1,
   0,0,0,0,0,1,0,0,1,0,0,1,0,1,0,1,1,1,1,1,0,0,0,0,1,1,0,
   0,0,0,0,1,0,0,1,0,1,1,1,1,1,1,0,0,0,0,1,1,0,1,1,1,1,1,
   0,0,1,0,0,1,1,1,0,0,0,1,1,1,1,1,1,0,0,0,1,1,0,1,1,1,1,
   0,0,0,0,0,1,0,0,1,0,1,1,1,0,1,1,1,1,1,1,0,0,0,1,1,0,1,
   0,0,0,0,0,1,0,0,1,0,1,1,1,0,0,0,1,1,0,1,1,1,1
```

```
AllGoldSequences::usage = "AllGoldSequences[PrimitivePoly] e.g. AllGoldSequences[
    {1,0,0,3,2,3}] = {{0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,1,0,1,1,1,0,
    0,0,1,1,1,0,0,1,1},{1,0,0,0,0,1,0,0,1,1,1,1,1,0,1,1,0,0,1,0,1,1,
    0,1,1,1,0,0,1,0,0,1},{0,1,0,0,0,0,1,0,0,0,0,1,1,1,0,0,0,0,1,0,0,
    1,0,0,1,0,1,1,0,1,1,1,0},{1,1,0,0,0,1,1,0,1,1,0,1,0,1,1,1,1,1,1,0,1,
    0,1,0,1,0,0,0,1,1,0,0},{0,0,1,0,0,1,0,1,1,1,1,0,1,1,1,0,1,1,0,1,0,1,
    1,0,0,0,1,1,0,1,1,1,1},{1,0,1,1,0,0,0,0,1,1,0,1,0,1,1,0,1,1,0,1,0,1,
    1,0,0,0,0,0,0,0,0,1,1,1,0,1},{0,1,1,0,0,1,1,1,1,1,0,0,0,0,0,0,1,1,
    0,1,0,1,0,0,1,0,1,1,1,1,0,1,0},{1,1,1,0,0,0,1,1,0,0,1,1,0,0,1,1,0,0,
    0,1,0,1,1,1,0,0,0,1,1,0,0,0,0,0,0},{0,0,0,1,0,0,1,1,0,0,1,1,0,0,1,
    1,0,0,0,1,1,1,0,1,0,1,0,0,1,1,0,0,1},{1,0,0,1,1,1,0,1,0,1,1,0,0,0,
    0,0,1,1,1,1,1,1,1,1,1,0,1,0,1,1,0,0},{0,0,0,0,0,0,0,1,1,1,1,1,1,1,
    0,0,0,0,0,0,0,0,1,0,1,1,1,1,1,1,1,0},{0,0,1,0,1,1,1,0,0,1,1,1,1,0,
    0,1,0,1,0,1,0,0,1,0,1,1,1,0,1,0,0,1},{1,0,1,1,0,0,1,1,0,1,1,1,1,0,
    0,1,0,0,1,1,1,0,1,1,0,0,1,1,1,1,1,1},{0,1,1,1,0,1,0,1,0,1,1,1,1,0,
    0,1,1,0,1,1,1,0,0,1,1,0,1,1,1,0,0,0},{1,1,1,1,1,0,0,0,0,1,1,0,1,1,
    1,1,1,1,1,1,1,1,1,1,0,0,1,1,1,1,1,0},{0,0,0,0,1,0,0,0,1,1,1,1,1,1,
    0,1,0,0,1,0,0,1,0,1,1,1,1,1,1,1,1,1},{1,1,0,
    0,1,1,1,1,1,1,1,0,0,1,1,0,1,1,1,0,0,0,0,0,0,1,1,1,1,0,1,1,1,1,1},{0,0,
    1,0,1,1,0,0,1,1,0,1,1,1,1,1,1,1,1,1
```

```
RingDivision[ModuloRing_][{Denominator_, Numerator_}] :=
 Module[{x1, x2, x3, x4, x5, x6, Qx, Nx},
  Clear[x1];
  x1[i_, j_] := Mod[i j, ModuloRing];
  x2 = Table[x1[Denominator[[1]], j], {j, 0, ModuloRing - 1}];
  x3 = Complement[Table[i, {i, 0, ModuloRing - 1}], x2];
  Qx = {};
  Nx = Numerator;
  NoelUpdate :=
    Module[{ux1, ux2, ux3, ux4},
    If[MemberQ[x3, Nx[[1]]], Throw[{Qx, Nx, Denominator, False}]];
    ux1 = Take[Nx, Length[Denominator]];
    ux2 = Position[x2, ux1[[1]] // Flatten // (#[[1]] - 1) &;
   ux3 = Mod[ux1 - ux2 Denominator, ModuloRing];
    Qx = Join[Qx, {ux2}];
   ux4 = Drop[Nx, Length[Denominator]];
   Nx = Join[Rest[ux3], ux4]];
 Catch[Table[ NoelUpdate, {i, 1, Length[Numerator] - Length[Denominator] + 1}];
   {Qx, Nx, Denominator, True}]
  ]

DropLeadingZeros[DenominatorSeq_] :=
  Module[{x1},
  x1[{}] := {};
  x1[DS_] /; DS[[1]] === 0 := x1[Rest[DS]];
 x1[DS_] := DS; x1[DenominatorSeq]
 ]

PossibleDivisors[ModuloRing_][Order_] :=
    Module[{x1, x2, x3, x4},
  x1 = Table[{x2[i], 0, ModuloRing - 1}, {i, 0, Order}];
  x3 = Table[x2[i], {i, 0, Order}];
x4 = Apply[Table, Sequence[Join[{x3}, x1]]] // Flatten[#, Order]& // Rest;
Map[DropLeadingZeros, x4] ]

PolynomialMultiplication[ModuloRing_][poly1_, poly2_] :=
    Module[{x1, x2, x3, x4, x5, x6, x7, x8, x9},
    x1 = Length[poly1] + Length[poly2] - 1;
    x2 = Table[x3^i, {i, 0, x1 - 1}];
    x4 = (Take[x2, Length[poly1]] // Reverse) . poly1;
    x5 = (Take[x2, Length[poly2]] // Reverse) . poly2;
    x6 = x5 x4;
    x7 = Table[Coefficient[x6, x3^(x1 - i - 1)] // Mod[#, ModuloRing]&, {i, 0, x1 - 2}];
    x8 = Join[x7, {Mod[x6 /. x3 -> 0, ModuloRing]}] // DropLeadingZeros]
```

We define polynomial multplication, modulo a primitive polynomial

```
ModuloMultplication[ModuloRing_][PrimitivePolynomial_][poly1_, poly2_] :=
 Module[{x1, x2, x3, x4},
   x1 = PolynomialMultiplication[ModuloRing][poly1, poly2];
  x2 = RingDivision[ModuloRing][{PrimitivePolynomial, x1}];
 x2[[2]] // DropLeadingZeros]

RingPower[RingModulo_][PrimitiveElement_][Element_] :=
  Module[{x1, st, NoelUpdate},
  x1 = Length[PrimitiveElement] - 1;
  st = {1};
  NoelUpdate :=
   st = ModuloMultplication[RingModulo][PrimitiveElement][ st, Element];
  Table[NoelUpdate, {i, 1, 2^x1 - 1}]]
```

```
CyclicMultiplativeGroup[ModuloRing_][PrimitivePolynomial_] :=
 Module[{x1, x2, x3, x4, x5, x6, x7, x8, x9},
  x1 = (Length[PrimitivePolynomial] + 1) / 2 // Floor;
  x2 = PossibleDivisors[ModuloRing][x1];
x3 = Map[RingDivision[ModuloRing][{#, PrimitivePolynomial}]&, x2];
x4 = Select[x3, #[[4]] == True &];
 x5 = Select[x4, #[[2]] == {1} &] // Transpose // Join[#[[1]], #[[3]]]&;
 x6 = Select[x4, #[[2]] == {ModuloRing - 1} &] // Transpose // Join[#[[1]], #[[3]]]&;
x7 = Join[{{1}, {ModuloRing - 1}}, x5, x6] // Union;
x8 =   Map[ModuloMultplication[ModuloRing][PrimitivePolynomial][#, #]&, x7] // Union;
x9 =  RingPower[ModuloRing][PrimitivePolynomial][x8[[2]]] // RotateRight  ]


ZeroSequences[ModuloRing_][Order_] :=
   Module[{x1, x2, x3, x4},
  x1 = Table[{x2[i], 0, 1}, {i, 1, Order }];
 x3 = Table[ModuloRing / 2 x2[i], {i, 1, Order }];
x4 =  Apply[Table, Sequence[Join[{x3}, x1]]] // Flatten[#, Order - 1]& ]

ZeroPad[Order_][Element_] :=
 Module[{x1, x2, x3},
  x1 = Table[0, {i, 1, Order - Length[Element]}];
  x2 = Join[x1, Element]]

TwoAdicExpansion[ModuloRing_][PrimitivePolynomial_] :=
 Module[{x1, x2, x3, x4, x5, x6, x7, x8, x9},
  x1 = Length[PrimitivePolynomial] - 1;
 x2 = CyclicMultiplativeGroup[ModuloRing][PrimitivePolynomial];
 x3 = Map[ZeroPad[x1], x2];
 x4 = Join[{Table[0, {i, 1, x1}]}, x3];
 x5 = Function[x, Mod[x[[1]] + 2 x[[2]], ModuloRing]];
  x6 = Table[{x4[[i]], x4[[j]], x5[{x4[[i]], x4[[j]]}] // DropLeadingZeros},
     {i, 1, Length[x4]}, {j, 1, Length[x4]}] //
    Flatten[#, 1]&]

σ[ModuloRing_][PrimitivePolynomial_][TwoadicElement_] :=
   Module[{x1, x2, x3, x4, x5, x6, x7},
  {x1, x2, x3} = TwoadicElement;
  x4 = Length[ PrimitivePolynomial] - 1;
  x5 = ModuloMultplication[ModuloRing][PrimitivePolynomial][x1, x1] // ZeroPad[x4];
x6 =  ModuloMultplication[ModuloRing][PrimitivePolynomial][x2, x2] // ZeroPad[x4];
Mod[2 x6 + x5, ModuloRing]]

AutomorphismSigma[ModuloRing_][PrimitivePolynomial_] :=
 Module[{x1, x2, x3, x4, x5, x6, x7, x8, x9},
  x1 = TwoAdicExpansion[ModuloRing][PrimitivePolynomial];
x2 = Map[σ[ModuloRing][PrimitivePolynomial], x1];
x3 = Map[ZeroPad[Length[PrimitivePolynomial] - 1], Transpose[x1][[3]]];
 {x3, x2} // Transpose]

UnitsRing[ModuloRing_][PrimitivePolynomial_] :=
 Module[{x1, x2, x3, x4},
  x1 = AutomorphismSigma[ModuloRing][PrimitivePolynomial];
x2 = x1 // Transpose // #[[1]]&;
x3 =  x2 // Mod[2 #, 4]& // Union;
 Complement[x2, x3]]

UnitsRing[4][{1, 0, 0, 3, 2, 3}];

τ[Order_][σ_][x_] := (NestList[σ, x, Order - 2] // Apply[Plus, #]&)

TraceRepresentation[ModuloRing_][PrimitivePolynomial_] :=
 Module[{x1, x2, x3, x4, x5, x6, x7, x8, x9},
  x1 = Length[PrimitivePolynomial] - 1;
  x2 = AutomorphismSigma[ModuloRing][PrimitivePolynomial];
  Map[{x3[Evaluate[#[[1]]]] := Evaluate[#[[2]]]}&, x2];
 x4 = Map[{#, τ[x1][x3][#]}&, Transpose[x2][[1]]] // Mod[#, ModuloRing]&

]
```

```
TupleRepresentation[ModuloRing_][PrimitivePolynomial_] := Module[{x1, x2, x3},
  x1 = Length[PrimitivePolynomial];
  x2 = NestList[ModuloMultiplication[ModuloRing][PrimitivePolynomial][{1, 0}, #]&,
    {1}, 2^(x1 - 1) - 2];
  x3 = Map[ZeroPad[x1 - 1], x2]]
```

This algorithm does not always yield the correct value for the Minimum polynomial.

```
MinimumPoly[ModuloRing_][PrimitivePolynomial_][rootPower_][var_] :=
  Module[{x1, x2, x3, x4, x5, x6, x7, x8, x9, x10, x11, x12, x13, x14, x15, tom},
    x1 = Length[PrimitivePolynomial] - 1;
   x2 = 2^x1 - 1;
x3 = NestList[Mod[# + #, x2]&, rootPower, x2];
x4 = Position[x3, rootPower] // (#[[2, 1]] - 1)&;
 x6 = Take[x3, x4] ;
  x7 = TupleRepresentation[ModuloRing][PrimitivePolynomial];
x8 = Table[i, {i, 0, x2 - 1}];
 x9 = (x8, x7} // Transpose;
  x10 = Map[(var ^ tom[#])&, x6] // Apply[Times, #]& // Expand;
  rule = tom[n_] tom[m_] -> tom[n + m];
  x11 = x10 //. rule;
 x12 = Table[ Coefficient[ x11, var, x1 - i], {i, 0, x1}];
 Map[(tom[Evaluate[#[[1]]]] := Evaluate[#[[2]]])&, x9];
 tom[n_] := Evaluate[ tom[Mod[n, x2]]]; Mod[
x12, ModuloRing]]

GraeffeMethod[GF2minpoly_] :=
  Module[
   {x1, x2odd, x2even, x3odd, x3even, x4even, x4odd, x5, x6even, x6odd, x7, x8, x9},
   x1 = Length[GF2minpoly];
   x2odd = Table[{i, 0}, {i, 1, (x1 + 1) / 2 // Floor}] // Flatten // Take[#, -x1]&;
   x2even = Table[{0, i}, {i, 1, (x1 + 1) / 2 // Floor}] // Flatten // Take[#, -x1]&;
 x3even = GF2minpoly x2even // DropLeadingZeros;
  x3odd = GF2minpoly x2odd // DropLeadingZeros;
x4even = PolynomialMultiplication[4][x3even, x3even];
x4odd = PolynomialMultiplication[4][x3odd, x3odd] ;
 x5 = Max[{Length[x4even], Length[x4odd]}];
 x6even = ZeroPad[x5][x4even];
  x6odd = ZeroPad[x5][x4odd];
x7 = x6even - x6odd;
 x8 = (x7 Sign[x7[[1]]]) // Mod[#, 4]&;
  Join[x8, {0}] // Partition[#, 2]& // Transpose[#][[1]]& ]

SequenceGenerator[ModuloRing_][PrimitivePoly_][InitialCond_][SeqLength_] :=
 Module[{x1, x2State, x3Update, x4},
  x1 = Mod[-Rest[PrimitivePoly], ModuloRing];
 x2State = InitialCond;
 x3Update := Module[{}, x4 = Last[x2State];
   x2State = Join[{Mod[x2State . x1, ModuloRing]}, Drop[x2State, -1]]; x4];
  (Table[x3Update, {i, 1, SeqLength}], x2State) ]

InitialConditions[PrimitivePoly_] := Module[{x1, x2, x3, x4, x5J, x6, x7},
  x1 = Length[PrimitivePoly] - 1;
   x2 = Table[{x3[i], 0, 1}, {i, 1, x1}]; x4 = Table[x3[i], {i, 1, x1}];
  x5J = (Table @@ Sequence[Join[{x4}, x2]]) // Flatten[#, x1 - 1]&;
x6 = Mod[x5J[[1]] + 2 x5J[[2]], 4];
x7 = Map[Mod[x5J[[2]] + 2 #, 4]&, x5J];
Join[x7, {x6}]]

GoldSequence[PrimitivePoly_][InitCondNumber_] := Module[{x1, x2, x3, x4},
  x1 = Length[PrimitivePoly] - 1;
  x2 = InitialConditions[PrimitivePoly][[InitCondNumber]];
x3 = SequenceGenerator[4][PrimitivePoly][x2][2^x1 - 1];
x4 = x3[[1]] /. {0 → 0, 1 → 0, 2 → 1, 3 → 1} ]

AutocorrelationSequence[GoldSequence_] :=
   Module[{x1, x2State, x3Update, x4},
    x1 = GoldSequence /. 0 → -1 ;
x2State = x1;
x3Update := Module[{}, x4 = x1 . x2State; x2State = RotateRight[x2State]; x4];
 Table[x3Update, {i, 1, Length[GoldSequence]}] ]
```

```
SpecifiedGoldSequences[PrimitivePoly_][NumberList_] :=
 Module[{x1, x2, x3, x4},
  x1 = Length[PrimitivePoly] - 1;
x2 =  NumberList /. Last → 2^x1 + 1;
 Map[GoldSequence[PrimitivePoly][#]&, x2] ]

AllGoldSequences[PrimitivePoly_] :=
 Module[{x1, x2, x3, x4},
  x1 = Length[PrimitivePoly] - 1;
x2 =  Table[i, {i, 1, 2^x1 + 1}];
 Map[GoldSequence[PrimitivePoly][#]&, x2] ]

CrosscorrelationSequence[{GoldSeq1_, GoldSeq2_}] :=
    Module[{x1, x2State, x3Update, x4},
     x1 = GoldSeq1 /. 0 → -1;
 x2State = (GoldSeq2 /. 0 → -1);
 x3Update := Module[{}, x4 = x1 . x2State; x2State = RotateRight[x2State]; x4];
  Table[x3Update, {i, 1, Length[GoldSeq1]}] ]

End[]

EndPackage[]
```

```
BeginPackage["LaurentFunctions`"]

T::usage = "This is the symbol period"

BT::usage = "This is the usual product"

h::usage = "This is the raw gaussian pulse"

Θ::usage = "This denotes modulation index Pi"

ψ::usage = "ψ[L,t] is Laurents function"

hFiltered::usage = "This is the filtered gaussian pulse"

PhaseAngle::usage = "This function takes some time to calculate. The following
    code will draw a graph of the functionPhaseFunction[t_] =N[ φL,t];
phasepoints =Table[{t,PhaseFunction[t T]}//N,{t,0,L ,1/40}];
ListPlot[phasepoints,PlotJoined -> True]"

PhaseAngleFast::usage = "PhaseAngleFast[L][t] speeds up the calculation
    of PhaseAngle by calculating PhaseAngle[L][t] with L numeric and t
    symbolic. It takes some time to calculate."

S::usage = "S = Sin[Θ]"

J::usage = "J = e^(i Θ)"

C::usage = "C = Cos[Θ]"

M::usage = "M = 2^(L-1)"

ModulationIndex::usage = "ModulationIndex = h"

LaurentS::usage = "LaurentS[L][n][t] = Sin[ ψ[L,t + n T]]/ S"

LaurentC::usage = "LaurentC[L][K][t] is Laurents C(K,t)"

AlphaKI::usage = "AlphaKI[LL][K,i] is Laurents α(K,i)"

LaurentLK::usage = "LaurentLK[L][K] gives the support of C(K,t)"
```

The start of Modulator Definitions

```
ANKInitialStateSetUp::usage =
 "ANKInitialStateSetUp[L][K][InitBitSeq,AccumulatedPhase] sets up the sequence
   of prior states of A(K,n) that the modulator went thgough to get to the
   constelation point specified by  AccumulatedPhase which is really A(0,0)"

AKN::usage = "AKN[L][K][{State,AccumulatedPhase}] defines A(K,n) in terms of A(0,n)"

ModulatingPulse::usage =
 "The Pulse is assumed to have the following structure Pulse[L][K][t]"

NumberOfCurves::usage = "The number of pulses used by the modulator"

SamplingInterval::usage =
 "SamplingInterval is the interval between samples of the output of the modulator"

InitialState::usage = "InitialState is the set of bits that are assumed
    to be present before i.e {a(-1),a(-2), ---}"

StartingQuadrant::usage = "StartingQuadrant = A(0,-1)and is a number"
```

```
Modulator::usage = "Modulator[L][BitSeq,Opts] assumes the following
   default options StartingQuadrant → 0,InitialState → Table[1,{4,
   1,20}],SamplingInterval → T/32,NumberOfCurves → 4,ModulatingPulse→
   LaurentC . The Pulse is assumed to have the following structure Pulse[L][K][t]"
```

Start of the Receiver Functions

```
FiltPulse::usage = "The default pulse and is called by FiltPulse[L][K][t]"

SyncSample::usage = "Given that the sampling interval is T/32,then sync
   sample has rang -16 to 16 and this moves the point used to demodulate "

Receiver::usage = " Receiver[L][InputSeq,Opts] assumes the following
   default options {StartingQuadrant→0,InitialState→{1,1,1,1,1,1,1,1,
   1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},SamplingInterval -> T/32,ModulatingPulse →
   FiltPulse,SyncSample -> 0,NumberOfCurves -> 2   "

ReceiverProper::usage = "ReceiverProper[L][ StartingQuadrant, InitialState,
   SamplingInterval,ModulatingPulse,SyncSample,NumberOfCurves,InputSeq]
   is called by Receiver after all the options have been resolved"

Begin["`Private`"]
```

$$\sigma := \frac{\sqrt{\text{Log}[2]}}{2\,\pi\,\text{BT}}$$

$$h[t\_] := \frac{\text{Exp}\left[-\frac{t^2}{2\,\sigma^2\,T^2}\right]}{\sqrt{2\,\pi}\,\sigma\,T}$$

Ideally we would and did define the effect of the convolution of h[t] by the formula below. However, this version of *Mathematica* gives an error.

$$h_{fil}[t\_] := \text{Release}\left[\text{Module}\left[\{\tau\},\ \int_{-\frac{T}{2}}^{\frac{T}{2}} \frac{h[t-\tau]}{T}\,d\tau\right]\right]$$

```
hFiltered[PulseWidth_][t_] := Module[{x1, x2, x3}, x1 =
```

$$(N[\#1,\ 40]\&)\left[\text{Table}\left[\{t1,\ \int_{-\frac{T}{2}}^{\frac{T}{2}} \frac{h[t1-\tau]}{T}\,d\tau\},\ \{t1,\ -\frac{\text{PulseWidth}}{2},\ \frac{\text{PulseWidth}}{2},\ \frac{T}{20}\}\right]\];$$

```
  x2 = Interpolation[x1]; x2[t]]
```

```
θ := N[ModulationIndex π]

C := Cos[θ];
S := Sin[θ];
J := (e^{iθ} // Chop);
M := 2^{L-1};

PhaseAngle[L_][t_] /; t ≤ 0 := 0

PhaseAngle[L_][t_] /; t ≥ L T := θ

PhaseAngle[L_][t_] :=
 PhaseAngle[L][t_] = Module[{x1, x2, x3, x4, x5, x6}, x1 = hFiltered[3 L T][t1 - LT/2];
   x2 = Table[{t2, θ ∫_{-LT}^{t2} Evaluate[x1] dt1}, {t2, 0, L T, T/100}]; Interpolation[x2][t]]
```

We need to ensure that we always culculate PhascAngle with t symbolic first to do the calculation only once

```
PhaseAngleFast[L_][t_] := Module[{x1}, x1[tt_] = PhaseAngle[L][tt]; x1[t]];
```

```
ψ[L_, t_] /; 0 < t < LT := PhaseAngleFast[L][t]

ψ[L_, t_] /; 2LT > t ≥ LT := 8 - PhaseAngleFast[L][t - LT]
```

We need to put this to avoid the function being extrapolated

```
ψ[L_, t_] /; ! (0 < t < LT) && ! (2LT > t ≥ LT) := 0

LaurentS[L_][n_][t_] := Sin[ψ[L, t + nT]] / 8

AlphaKI[LL_][K_, i_] /; (0 < i < LL) && (0 <= K < 2^(LL-1)) :=
 Module[{x1, x2, x3, KNum}, x1 := {x2 = Mod[KNum, 2]; KNum = (KNum - x2)/2; x2};
  KNum = K; x3 = Table[x1, {ii, 0, LL - 1}]; x3[[i]]]

LaurentLK[L_][K_] :=
 Module[{x1}, x1 = Table[L (2 - AlphaKI[L][K, ii]) - ii, {ii, 1, L - 1}]; Min[x1]]

LaurentC[L_][K_][t_] /; 0 <= K < 2^L :=
 LaurentS[L][0][t] ∏_{i=1}^{L-1} LaurentS[L][ii + L AlphaKI[L][K, ii]][t]
```

The start of the modulator function

```
ANKInitialStateSetUp[L_][K_][InitBitSeq_, AccumulatedPhase_] :=
      Module[{x1, x2, x3, x4, x5, acuphase, initbitseq},
  initbitseq = InitBitSeq;
  acuphase = AccumulatedPhase;
   UpdateSeq :=
   Module[{}, x1 = acuphase - Sum[initbitseq[[i]] AlphaKI[L][K, i], {i, 1, L - 1}];
    acuphase = acuphase - First[initbitseq]; initbitseq = Rest[initbitseq]; x1],
  Table[ UpdateSeq, {i, 1, LaurentLK[L][K]}]]

AKN[L_][K_][{State_, AccumulatedPhase_}] :=
 AccumulatedPhase - Sum[State[[i + 1]] AlphaKI[L][K, i], {i, 1, L - 1}]

Options[Modulator] := {StartingQuadrant → 0, InitialState → Table[1, {i, 1, 20}],
  SamplingInterval → T / 32, NumberOfCurves → 4, ModulatingPulse → LaurentC}

Modulator[L_][BitSeq_, Opts___] :=
 Module[
   {x1, x2, x3, x4, x5, x6, state, AccumulatedPhase, seq, AKNState, Curves, Pulse},
x1 = SamplingInterval /. {Opts} /. Options[Modulator];
state = InitialState /. {Opts} /. Options[Modulator];
x3 = StartingQuadrant /. {Opts} /. Options[Modulator];
x4 = SamplingInterval /. {Opts} /. Options[Modulator];
Pulse = ModulatingPulse /. {Opts} /. Options[Modulator];
Curves = (NumberOfCurves /. {Opts} /. Options[Modulator]) - 1;
AccumulatedPhase = x3;
seq = BitSeq;
Table[
    AKNState[K] = ANKInitialStateSetUp[L][K][state, AccumulatedPhase], {K, 0, Curves}];
x5 := Module[{}, state = Join[{First[seq]}, Drop[state, -1]];
AccumulatedPhase = AccumulatedPhase + First[seq];
seq = Rest[seq];
Table[AKNState[K] = Join[{AKN[L][K][{state, AccumulatedPhase}]},
        Drop[AKNState[K], -1]], {K, 0, Curves}];
x6[t_] = Sum[Sum[ (J)^AKNState[K][[i+1]] Pulse[L][K][t + i T],
    {i, 0, LaurentLK[L][K] - 1}], {K, 0, Curves}];
Table[x6[t], {t, 0, T - x4, x4}]];
 Table[x5, {kk, 1, Length[BitSeq]}] // Flatten]

Options[Receiver] := {StartingQuadrant → 0,
  InitialState → {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}, SamplingInter
  ModulatingPulse → FiltPulse, SyncSample -> 0, NumberOfCurves -> 2};
```

```
Receiver[L_][InputSeq_, Opts___] :=
  Module[{x1, x2, x3, x4, x5, x6},
   x1 = StartingQuadrant /. {Opts} /. Options[Receiver];
   x2 = InitialState /. {Opts} /. Options[Receiver];
   x3 = SamplingInterval /. {Opts} /. Options[Receiver];
   x4 = ModulatingPulse /. {Opts} /. Options[Receiver];
   x5 = SyncSample /. {Opts} /. Options[Receiver];
   x6 = NumberOfCurves /. {Opts} /. Options[Receiver];
ReceiverProper[L][x1, x2, x3, x4, x5, x6, InputSeq]]

ReceiverProper[L_][StartingQuadrant_, InitialState_, SamplingInterval_,
  ModulatingPulse_, SyncSample_, NumberOfCurves_, InputSeq_] :=
    Module[{x1, sgn, ReceivedSeq, ExpectedValue, seq, ReceiveNext, D, N},
   x1 = T / SamplingInterval;
   ReceivedSeq = Partition[InputSeq, x1] // Transpose // #[[SyncSample + 1]]&;
   ExpectedValue =
    ModulatingPulse[L][0][(LaurentLK[L][0] / 2) T + SyncSample SamplingInterval];
   N = StartingQuadrant;
   sgn = 0;
   D = {};
   seq = ReceivedSeq;
   ReceiveNext :=
   Module[{x1, x2},
     x1 = ( (-1)^sgn J^N First[seq]) // Im;
    If[Abs[x1 - ExpectedValue] <= Abs[x1 + ExpectedValue],
      D = Join[ D, {1}];  N = N + 1,  D = Join[ D, {-1}];  N = N - 1 ];
     seq = Rest[seq]; sgn = Mod[sgn + 1, 2]];
 Table[ReceiveNext, {i, 1, Length[ReceivedSeq]}];
  D]

End[]

EndPackage[]
```